

**Pascal Pons**[Follow](#)

SOLVING CONNECT FOUR

1. Introduction
2. Test protocol
3. MinMax algorithm
4. Alpha-beta algorithm
5. Move exploration order
6. Bitboard
7. Transposition table
8. Iterative deepening
9. Anticipate losing moves
10. Better move ordering
11. Optimized transposition table
12. Lower bound transposition table

Part 7 – Transposition Table

When you explore the game tree, you end up analysing several times the same positions that can be reached from different combination of moves. You can save time by caching the outcome of previous computation to avoid to re-compute the score of a position you already explored in the past.

The idea is similar to [dynamic programming](#): you trade computational time against memory space. Unfortunately you won't be able to store all the positions and you have to come up with some strategies to optimize which positions to keep in memory.

Transposition Table

Efficient transposition tables will try to keep the positions that will help you saving the most computation time. It can be either frequent positions or positions that take a lot of time to compute. We implement a strategy favoring storage of the most recently explored nodes.

Our very simple transposition table is mainly a hash table with fixed storage size that overriding previous entries (no collision management). Keeping the most recent positions is a good and simple strategy because recently explored nodes are often close to the current position. This strategy increases the probability to have a positive hit in your cache.

Alpha-beta outcome can be a lower or an upper bound of the actual score. In this version, we will only cache the upper bound results. Storing both upper and lower bound complexifies the implementation and does not bring in practice much improvement. We will consider this smaller optimization later.

Implementation

Our simple Transposition Table implementation supports 56-bit keys and 8-bit values, making 64-bit entries. Here is the simplified interface we need:

```
class TranspositionTable {
public:

/**
 * Store a value for a given key
 * @param key: 56-bit key
 * @param value: non-null 8-bit value. null (0) value are used to encode missing data.
 */
void put(uint64_t key, uint8_t val);

/**
 * Get the value of a key
 * @param key
 * @return 8-bit value associated with the key if present, 0 otherwise.
 */
uint8_t get(uint64_t key) const;
```

Check the [full detailed implementation](#) of this simple interface.

Implementation in the negamax function is straitforward. We simply check at the beginning of the negamax function if the current position is already in the transposition table. If the value is present in the cache it may allow to narrow the exploration window and in some cases, generate an immediate pruning of the exploration. At the end of the negamax function, when we have an upper bound, we simply store the value in the Transposition table.

Full [source code](#) corresponding to this part.

Benchmark

For this benchmark, we took a 64MB transposition table (same size as Fhourstone reference). It reduces significantly the number of explored node, especially when a deeper search is needed. We are now able to solve most of the beginning of game test cases in a reasonbale time:


Solver	Test Set name	mean time	mean nb pos	K pos/s
Transposition Table (strong solver)	End-Easy	6.531 µs	92.84	14,220
Transposition Table (strong solver)	Middle-Easy	5.594 ms	207,900	37,170
Transposition Table (strong solver)	Middle-Medium	52.45 ms	1,731,000	33,000
Transposition Table (strong solver)	Begin-Easy	4.727 s	156,400,000	33,090
Transposition Table (strong solver)	Begin-Medium	8.200 s	306,100,000	37,330
Transposition Table (strong solver)	Begin-Hard	N/A	N/A	N/A

Solver	Test Set name	mean time	mean nb pos	K pos/s
Transposition Table (weak solver)	End-Easy	5.155 μ s	68.69	13,320
Transposition Table (weak solver)	Middle-Easy	1.072 ms	28,750	26,830
Transposition Table (weak solver)	Middle-Medium	23.79 ms	752,300	31,620
Transposition Table (weak solver)	Begin-Easy	1.610 s	52,840,000	32,830
Transposition Table (weak solver)	Begin-Medium	1.762 s	63,930,000	36,280
Transposition Table (weak solver)	Begin-Hard	N/A	N/A	N/A

Tutorial plan

SOLVING CONNECT FOUR

1. Introduction
2. Test protocol
3. MinMax algorithm
4. Alpha-beta algorithm
5. Move exploration order
6. Bitboard
7. Transposition table
8. Iterative deepening
9. Anticipate losing moves
10. Better move ordering
11. Optimized transposition table
12. Lower bound transposition table

 **Updated:** April 24, 2017

[Previous](#)[Next](#)